

队列(Queue)

我们之前说到了栈，它是一种比较高效的数据结构，遵循 先进后出(LIFO, last-in-first-out) 的原则。而今天我们要讨论的**队列**，它也是一种特殊的列表，它与栈不同的是，队列只能在队尾插入元素，在队首删除元素，就像我们平时排队买票一样~

队列用于存储按顺序排列的数据，遵循 先进先出(FIFO, First-In-First-Out) 的原则，也是计算机常用的一种数据结构，别用于很多地方，比如提交给操作系统的一系列进程，打印池任务等。

同栈有点类似，队列的操作主要也是有两种：向队列中插入新元素和删除队列中的元素，即入队和出队操作，我们采用 enqueue 和 dequeue 两个方法。

除此之外，队列还有一些其他的操作，比如读取队首的元素，该操作仅返回对头元素并不将它从队列中删除，类似栈的 peek 方法；back 方法读取队尾的元素；toString 方法可以打印当前队列中所有的元素；clear 方法清空当前队列等。

| 类型 | 描述 |
|--------------|-------------|
| enqueue(方法) | 向队列末尾添加一个元素 |
| dequeue(方法) | 删除队列首的元素 |
| front(方法) | 读取队列首元素 |
| back(方法) | 读取队列尾的元素 |
| toString(方法) | 显示队列所有的元素 |
| clear(方法) | 清空当前队列 |
| empty(方法) | 判断队列是否为空 |

队列数据定义

我们定义好数据类型，可以通过JS中的数组去实现它。

队列的实现

```
1 //定义队列
2
3 function Queue(){
4     this.dataStore = [];
```

```
5     this.enqueue = enqueue;      //入队
6     this.dequeue = dequeue;      //出队
7     this.front = front;          //查看队首元素
8     this.back = back;            //查看队尾元素
9     this.toString = toString;    //显示队列所有元素
10    this.clear = clear;           //清空当前队列
11    this.empty = empty;           //判断当前队列是否为空
12 }
13
```

我们先来实现入队操作：

enqueue：向队列添加元素

```
1 //向队列末尾添加一个元素，直接调用 push 方法即可
2
3 function enqueue ( element ) {
4     this.dataStore.push( element );
5 }
6
```

因为JS中的数组具有其他语言没有的有点，可以直接利用 shift 方法删除数组的第一个元素，因此，出队操作的实现就变得很简单了。

dequeue：删除队首的元素

```
1 //删除队列首的元素，可以利用 JS 数组中的 shift 方法
2
3 function dequeue () {
4     if( this.empty() ) return 'This queue is empty';
5     else this.dataStore.shift();
6 }
7
```

我们注意的，上面我做了一个判断，队列是否还有元素，因为去删除空队列的元素是没有意义的，那么，我们就来看看 empty 方法是如何实现的。

empty: 判断队列是否为空

```
1 //我们通过判断 datastore 的长度就可知道队列是否为空
2
3 function empty(){
4     if( this.dataStore.length == 0 ) return true;
5     else return false;
6 }
7
```

我们先来看看测试一下这几个方法,

```
1 var queue = new Queue();
2
3 console.log( queue.empty() );          //true
4
5 //添加几个元素
6 queue.enqueue('Apple');
7 queue.enqueue('Banana');
8 queue.enqueue('Pear');
9
10 console.log( queue.empty() );          // false
11
```

现在, 我不知道谁在第一个, 谁是最后一个, 我们可以利用 front 和 back 方法分别来查看,

front: 查看队首元素

```
1 //查看队首元素, 直接返回数组首个元素即可
2
3 function front(){
4     if( this.empty() ) return 'This queue is empty';
5     else return this.dataStore[0];
6 }
```

back: 查看队尾元素

```
1 //查看队首元素，直接返回数组最后一个元素即可
2
3 //读取队列尾的元素
4 function back () {
5     if( this.empty() ) return 'This queue is empty';
6     else return this.dataStore[ this.dataStore.length - 1 ];
7 }
8
```

我们先看看对不对：

```
1 //查看队首元素
2 console.log( queue.front() ); // Apple
3 //查看队尾元素
4 console.log( queue.back() ); // Pear
5
6 //出队
7 queue.dequeue();
8
9 //查看队首元素
10 console.log( queue.front() ); // Banana
11 //查看队尾元素
12 console.log( queue.back() ); // Pear
13
```

没问题！现在，我想看看，总共有多少水果，toString方法来实现，

toString: 查看队列中所有元素

```
1 //查看对了所有元素，我这里采用数组的 join 方法实现
2
3 function toString(){
4     return this.dataStore.join('\n');
5 }
6
```

现在，你可以看看你还有什么水果没吃的了，

```
1 console.log( queue.toString() )    // Apple
2                                    // Banana
3                                    // Pear
4
```

我们就剩下一个 clear 方法了，如果你已经把所有水果都吃完了，那么你应该使用 clear 方法，

```
1 //清空当前队列，也很简单，我们直接将 dataStore 数值清空即可
2
3 function clear(){
4     delete this.dataStore;
5     this.dataStore = [];
6 }
7
```

至此，我们已经用JS实现了一个队列，怎么样，是不是觉得JS的数组超级好用，省去了不少麻烦，有木有！！

```
1 //清空队列
2
3 queue.clear();
4
```

```
5 console.log( queue.empty() );    // true
6
```

下面，我们利用队列来实现基数排序。

基数排序 (radix sort) 属于“分配式排序” (distribution sort)，它是透过键值的部份资讯，将要排序的元素分配至某些“桶”中，藉以达到排序的作用，基数排序法是属于稳定性的排序，其时间复杂度为 $O(n \log(r)m)$ ，其中 r 为所采取的基数，而 m 为堆数，在某些时候，基数排序法的效率高于其它的稳定性排序法。

先看一下基数排序的实现步骤（以两位数为例），需要扫描两次，第一次按个位数字进行排序，第二次按十位数字排序，每个数字根据对应的数值分配到不同的盒子里，最后将盒子的数字依次取出，组成新的列表即为排序好的数字。

1. 假设我们有一串数字，分别为 73, 22, 93, 43, 55, 14, 28, 65, 39, 81
2. 首先根据个位数字排序，放到不同的盒子里，如下图



第一次排序

3. 接下来将这些盒子中的数值重新串接起来，成为以下的数列：81, 22, 73, 93, 43, 14, 55, 65, 28, 39
4. 然后根据十位数字排序，再放到不同的盒子里，如下图



第二次排序

5. 接下来将这些盒子中的数值重新串接起来，整个数列已经排序完毕：14, 22, 28, 39, 43, 55, 65, 73, 81, 93

我们已经了解了基数排序的算法思想，接着我们要结合队列去实现它，一起来看看吧。

```
1 //基数排序
2
3 var queues = [];    //定义队列数组
4 var nums = [];      //定义数字数组
5
6 //选十个0~99的随机数进行排序
7 for ( var i = 0 ; i < 10 ; i ++ ){
8     queues[i] = new Queue();
9     nums[i] = Math.floor( Math.random() * 101 );
10 }
11
12 //排序之前
13 console.log( 'before radix sort: ' + nums );
14
15 //基数排序
16 distribution( nums , queues , 10 , 1 );
17 collect( queues , nums );
18 distribution( nums , queues , 10 , 10 );
19 collect( queues , nums );
20
21 //排序之后
22 console.info('after radix sort: ' + nums );
23
24 //根据相应的（个位和十位）数值，将数字分配到相应队列
25
26 function distribution ( nums , queues , n , digit ) { //digit表示个位或者十位的值
27     for( var i = 0 ; i < n ; i ++ ){
28         if( digit == 1){
29             queues[ nums[i] % 10 ].enqueue( nums[i] );
30         }else{
31             queues[ Math.floor( nums[i] / 10 ) ].enqueue( nums[i] );
```

```

32     }
33 }
34 }
35
36 //从队列中收集数字
37
38 function collect ( queues , nums ) {
39     var i = 0;
40     for ( var digit = 0 ; digit < 10 ; digit ++ ){
41         while ( !queues[digit].empty() ){
42             nums[ i++ ] = queues[digit].front();
43             queues[digit].dequeue();
44         }
45     }
46 }
47

```

我这里贴出两组测试的结果，大家自己动手实践一下。

```

1 //第一组测试
2
3 before radix sort: 23,39,2,67,90,41,47,21,98,13
4 after radix sort: 2,13,21,23,39,41,47,67,90,98
5
6 //第二组测试
7
8 before radix sort: 29,62,38,16,55,26,33,54,76,65
9 after radix sort: 16,26,29,33,38,54,55,62,65,76
10

```

至此，我们队列也就告一段落了，大家还可以往深入拓展，比如优先队列，循环队列等

作者：Cryptic

链接：<https://www.jianshu.com/p/1157aaccad36>

来源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。